# CoCoALib - Feature #209

## ReadExpr: input polynomials in CoCoALib

24 Jul 2012 17:17 - Anna Maria Bigatti

| | | | |
|---|---|---|---|
| **Status:** | Closed | **Start date:** | 24 Jul 2012 |
| **Priority:** | High | **Due date:** | |
| **Assignee:** | Anna Maria Bigatti | **% Done:** | 100% |
| **Category:** | New Function | **Estimated time:** | 30.00 hours |
| **Target version:** | CoCoALib-0.99532 | **Spent time:** | 30.75 hours |

**Description**

It would be nice if one could read polynomials from an input stream the way we write them, for example as

```
3*x^2*y-9*x^3*y^5
```

or perhaps even as expressions like

```
(x+1)*(y-1)
```

We have a powerful interpreter for cocoa-5, but it would be quite hard to extract from it the code for polynomial expressions.

Maybe we could have the first (3x^2y-9x^3y^5) without too much difficulty, but it's very tedious code to write.....

[originally requested by Bruns: email of 2012-07-24]

**2014-March** implemented as **ReadExpr(ring, expr)** / **ReadExprSemicolon(ring, expr)** (see ex-PolyInput2.C)

**Related issues:**

| | | |
|---|---|---|
| Related to CoCoALib - Feature #253: W.Bruns's wish list | **Closed** | **04 Oct 2012** |
| Related to CoCoA - Support #425: Osnabrueck 2014-01 | **Closed** | **27 Jan 2014** |
| Related to CoCoA-5 - Feature #7: Automatic mapping between (some) rings | **Resolved** | **20 Oct 2011** |
| Related to CoCoALib - Feature #913: Read input from SymbolicData database | **New** | **02 Aug 2016** |

### History

#### #1 - 30 Jul 2012 15:35 - John Abbott

JAA thinks polynomial input should accept expressions built from integer constants, symbols (with integer constant indexes), the four basic arithmetic operations (and exponentiation with integer constant exponent), and round brackets. Using * for multiplication should be obligatory.

We could consider recognising input of the form ***...*** where the ... represents an expression which does not use * for products.

Below is some of my reasoning for wanting to allow such a wide range of expressions.

1. The ability to read polynomials should work for polynomials in any ring, though I think we will likely want to exclude rings containing anonymous symbols (or at least polynomials containing terms involving anonymous symbols).
2. As an example we must allow input of values in QQ(a)[x] such as 1/(a+1); this requires both division and brackets!
3. It is unreasonable to expect the user to enter the terms of a polynomial in the correct order; analogously we should allow coefficients to be input in non-canonical form (*e.g.* 1/sqrt[2] instead of sqrt[2]/2)
4. Since division must be allowed for coefficients, we may as well allow it between polynomials (it would probably be quite tricky/artificial to write code which forbids it).

#### #2 - 31 Jul 2012 14:42 - John Abbott

We must decide how to handle white space inside RingElem inputs.

As a rule-of-thumb we should accept as input (at least) whatever CoCoALib produces as output. As an example the polynomial x+1 is actually printed as x +1 by CoCoALib (there is a space between x and +1).

We want the input syntax to be *readily readable* and also *convenient* whenever the user has to type in (small) polynomials by hand. The readability condition suggests forbidding spaces inside integer constants and inside names of indeterminates (but they are allowed anywhere else).

JAA suggests that TAB and SPACE be handled equivalently because there is usually no visual distinction between them when a file is displayed/printed.

Allowing spaces/tabs inside input elements probably makes it difficult to have more than one input element on a line (unless there is a visible separator). Think about the single input x-y and the two separate inputs x and -y. Because of this difficulty JAA suggests that we restrict to a single input element per line.

Inspired by the example above, we should **not allow** unescaped NEWLINEs inside an input element; otherwise it might be impossible to know when an input is finished: think about x on one line and -y on the next (or perhaps after several blank lines).

#### #3 - 31 Jul 2012 15:09 - John Abbott

The case of incorrect input which is syntactically correct seems to be easy:

- the input string is converted successfully into an expression tree
- evaluation of the expression tree fails (throwing an appropriate exception).

Two obvious reasons for the failure of evaluation are:

1. quotient cannot be computed (*e.g.* division by zero-divisor,...)
2. symbol has no image in the ring (*e.g.* misspelled head or index is wrong)

#### #4 - 31 Jul 2012 15:39 - John Abbott

In the previous note I suggest that *erroneous but syntactically correct* input is easy to handle. The case of *syntactically incorrect* input is less clear cut.

A simple and clear cut solution is to say that an input element is an entire line; however, this would be quite unnatural/inconvenient for reading matrices of ring elements.

The simple rule *read as much as possible that is syntactically correct* is problematic: it is hard to implement (as it would require the ability to back up arbitrarily far), also it is not so easy for a person to understand (*e.g.* the longest syntactically correct prefix may end long before the actual error). Here is an example to illustrate these points, note that the closing square bracket is missing:

```
1+x[1,2,3,4,5,...,999
```

JAA suggests the following compromise rule: **read the longest string which could be completed into a syntactically correct input** -- syntactically correct inputs are included, of course! If the string is not syntactically correct then an exception is thrown, otherwise the string is converted to an expression and evaluated (as in my previous note).

#### #5 - 31 Jul 2012 16:19 - John Abbott

Here are some sample inputs (which we can use for testing):

- - 1 the same as -1
- (-1) this is an expression, not an integer literal
- 2^64 this is an expression, not an integer literal
- 2^(64) syntax error after reading 2^
- 2^2^6 syntax error after reading 2^
- 1/2 syntax OK, evaluation OK if characteristic is not 2
- 1/-2 probably syntax error after reading 1/; what do you think?
- 1/0 syntax OK
- 1/x syntax OK, error if x is not invertible
- x*(1/x) syntax OK, error if x is not invertible
- x-x syntax OK, error if x is not in the ring
- 1/2/3 syntax OK or error??
- 1/2*x syntax OK or error?? (or warning?)
- 2x syntax OK **but** the expression read is just 2 -- the x is not read!
- 2,3 syntax OK **but** the expression read is just 2 -- next char is ,
- 2.3 syntax OK **either** this gives just 2 **or** it gives 23/10, **WHICH**? Do we also allow just **.3**
- xy syntax OK, the expression is the symbol xy
- x y syntax OK, the expression is the symbol x -- the next char to be read is y
- (x^2-1)/(x-1) syntax OK
- (x^2-1)*(x-1) syntax OK
- (x^2-1)(x-1) syntax OK, but expression read is just (x^2-1)
- x+-2 syntax error after reading x+
- x[1] syntax OK
- x[1+2] syntax error after reading x[1 because indices must be integer literals
- x[N] syntax error after reading x[

**Feel free to add further examples!**

#### #6 - 09 Oct 2012 15:28 - John Abbott

[following on from a discussion with Christof]

It does not seem to be possible to allow more than one input expression per line (unless there is an explicit separator): otherwise how would we distinguish the single expression x-y from the two expressions x and -y?

Some input expressions may be very long, and we should allow them to span several lines (rather than require that the input reside on a single very

long line -- perhaps several megabytes). Such a multi-line input requires an explicit end-of-expression marker; otherwise the next input line might continue the expression (*e.g.* by containing simply **+1**). The end-of-expression marker should be consumed when the expression is read.

On the other hand, many hand-typed inputs will be very short; and for this sort of input it would be most convenient if the newline character acted also as an end-of-expression marker. For short inputs it would be tedious and unnatural to require explicit end-of-expression markers (other than newline).

We want to allow easy entry of lists/vectors/matrices of polynomials; in this case the commas/brackets separating the successive entries would mark the end of the expressions (but would not be consumed by the reading of the expression). For instance, one could type the 3 element list [x,y,z] without any extra markers for the ends of the 3 expressions. In this case we could also allow multi-line inputs because we know that there will be an explicit marker at the end of the expression (though we don't know precisely what it will be -- *e.g.* could be comma or close square bracket).

Summary:

- sometimes when reading an expression we want newline to be considered as an end-marker
- sometimes we want newline not to be considered an end-marker
- sometimes the user may want to say explicitly that newlines should be accepted in expressions (and the user must then state what will be the end-marker)

Christof pointed out that requiring that ignorable newlines be "escaped" is **not practical** if multiline input is to be copied using cut-and-paste.

Comments? Opinions? Ideas?

#### #7 - 16 Jan 2014 16:57 - John Abbott

Would it make sense to discuss this when we go to Osnabrueck? (in 10 days' time)

**Note: remember this KISS philosophy!**

#### #8 - 27 Jan 2014 12:13 - John Abbott

*- Status changed from New to In Progress*

*- % Done changed from 0 to 10*

Anna, Christof & John propose the following for an initial impl:

- input polys are terminated by newline
- the whole (rest of) line is read even if a syntax error occurs part way
- round brackets are allowed (at least for rational coeffs)
- it is probably easier to allow (x+1)^2 than forbid it
- decimal numbers would be handy, but perhaps not for the very first impl(?)
- whitespace is ignored (except inside an identifier or inside an integer literal)

Identifiers are strings of letters; they may have integer literal indexes between square brackets, and separated by commas.

#### #9 - 30 Jan 2014 14:10 - Anna Maria Bigatti

*- Status changed from In Progress to Resolved*

*- Assignee set to Anna Maria Bigatti*

*- Priority changed from Normal to High*

*- Target version set to CoCoALib-0.9953*

*- % Done changed from 10 to 70*

Example program is done and working fully:
reads any kind expression with + - * / ^ ()

They key was John Abbott's suggestion for the design.
(neat! .... indeed I should have thought of it myself from my LP course)

#### #10 - 30 Jan 2014 17:02 - Anna Maria Bigatti

*- Status changed from Resolved to Feedback*

*- % Done changed from 70 to 90*

Official part of CoCoALib now.
Even with example in
example/ex-PolyInput2.C

(still missing: documentation)

#### #11 - 01 Mar 2014 10:06 - Anna Maria Bigatti

added RingElemInput in documentation (cvs-ed)
(work in progress)

#### #12 - 21 Mar 2014 14:27 - Anna Maria Bigatti

*- Target version changed from CoCoALib-0.9953 to CoCoALib-0.99532*

#### #13 - 21 Mar 2014 15:13 - Anna Maria Bigatti

**Warning!!**
Syntax for ReadExpr has changed: now the ring is the first argument (looking like RingElem(P, "x"))

#### #14 - 01 Apr 2014 18:54 - Anna Maria Bigatti

*- Status changed from Feedback to Closed*

*- % Done changed from 90 to 100*

#### #15 - 01 Apr 2014 18:55 - Anna Maria Bigatti

*- Estimated time set to 30.00 h*

#### #16 - 27 Aug 2014 18:38 - Anna Maria Bigatti

*- Subject changed from input polynomials in CoCoALib to ReadExpr: input polynomials in CoCoALib*

#### #17 - 02 Aug 2016 10:55 - John Abbott

*- Related to Feature #913: Read input from SymbolicData database added*